RoZ version 0.3 an environment

for the integration of UML and Z

Sophie Dupuy
Laboratoire Logiciels, Systèmes et Réseaux – IMAG
BP 72 – 38402 Saint-Martin d'Hères Cedex
Email : dupuy@imag.fr

1- Introduction to RoZ

1.1- Overview

RoZ is an environment to integrate UML ([UML97]) and Z ([SPI95]) notations. It is an extension of the Rational Rose tool which makes the transition between the UML world and the Z world. The Rose environment is used to build the UML models which are completed by formal annotations in Z. So the main point of RoZ is that all the modeling work (diagrams and annotations) should be realized in Rose to be the nearer to the usual working environment.

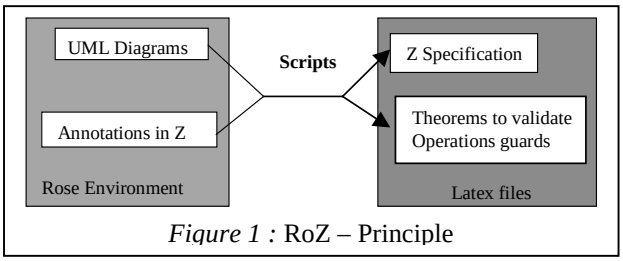


Figure 1 : RoZ – Principle

The RoZ tool proposes three functionalities :

- **Generation of Z specifications**

A complete Z specification can be generated from a UML class diagram and its annotations. The UML model is translated into Z formal skeletons which are completed by the annotations ([DUP99]). So the UML model supplies the formal specification structure while the annotations provide the details. The UML translation into Z is only made for a subset of UML class diagrams.

- **Generation of base operations**

The main idea is that for some concepts, there are operations which always exist. For instance, a class has always operations to modify its attributes. Moreover if the class is concrete, there are also operations to add or remove an instance to or from the class instances. Then for a class, the specification of these operations can be generated automatically.

- **Generation of theorems to validate operation guards**

From a subset of a UML class diagram, base operations and a Z formal specification can be generated automatically. Data must often respect constraints which are not taken into account by the code generated automatically and only implicitly taken into account in the specification. So it is interesting to ``design'' guards which will be evaluated before the operation execution and which can avoid to execute an operation violating the constraints. From the Z specification produced from the UML diagram, it is possible to prove that the proposed guards guarantee data integrity. So for each identified guard, a theorem is generated ([LED98]). The theorems demonstration uses a semi-automatic proof tool like Z-EVES ([SAA97]).

1.2- Prerequistes

Before you start using RoZ, you should be familiar with :

- UML ([UML97])

RoZ manages only the class diagram, in particular the notions of class, relationship (association, aggregation, composition), inheritance.

- Rose ([ROSE4.0,ROSE98])

You should know how to use Rose to construct your UML class diagrams.

- Z ([SPI95]) and its Latex format

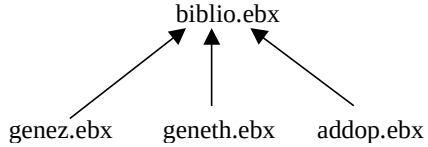
The annotations added to the UML diagrams should be written in the Z Latex format. So you should be familiar with this format.

The remainder of this document contains:

- an installing section which explains how modifying your Rose environment to enable the generation of Z specification;
- a guided tour that illustrates on a simple example the use of RoZ,
- a principles section which presents the basic principles of RoZ.

1.3- Installing

The three functionalities of RoZ has been implemented in three scripts : addop.ebx does the generation of base operations, genez.ebx allows to generate Z specification and geneth.ebx generates the theorems to validate operation guards. Moreover a file (biblio.ebx) gathers the common functions of the 3 previous scripts. So the files have the following architecture :



To install the 3 scripts in your environment, you can follow the instructions to extend Rose Menus given in the Rose ‘‘Extensibility Guide” (p7).

For **Unix version**, you have to:

1- put the scripts (addop.ebx, genez.ebx, geneth.ebx and biblio.ebx) into a directory
2- find the Rose menu file (for example, the menu file for Rose C++ is rose_cpp.mnu)
3- modify the Rose menu file to add the scripts to the main menu; add the following instructions at the end of the menu Tools declaration :

```
menu "Formal"
      {
      option "&Z Generation   Ctrl + G"
              {
               RoseScript $SCRIPT_PATH\genez.ebx
               }
      option "&Theorems Generation  "
              {
               RoseScript $SCRIPT_PATH\geneth.ebx
               }
      option "&Base Operations Generation  "
              {
               RoseScript $SCRIPT_PATH\addop.ebx
               }
      }
```

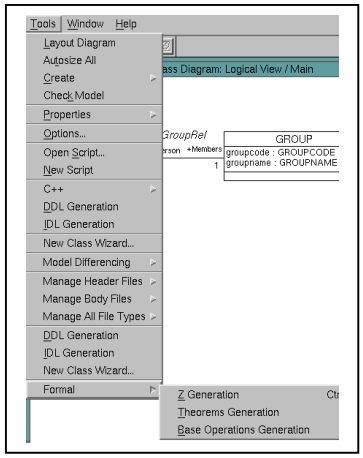
4- find YOUR .rose.ini file

5- in .rose.ini, add the new Rose menu file to the path
6- in Rose, in the ‘‘File” menu select ‘‘Edit Path Map”
7- select the symbol $SCRIPT_PATH and enter the path of the scripts genez.ebx, addop.ebx, geneth.ebx and biblio.ebx.
8- save the new path and exit Rose

For **Windows version**, you have to:

1. find the directory where you Rose scripts are: in Rose, in the ‘‘File” menu select ‘‘Edit Path Map”; select the symbol $SCRIPT_PATH and note the path of the scripts. If you do not have a $SCRIPT_PATH variable, choose a directory to put the scripts in and create a have a $SCRIPT_PATH that refers to this directory
2. put the scripts (addop.ebx, genez.ebx, geneth.ebx and biblio.ebx) into this directory
3. find the Rose menu file (for example, the menu file for Rose C++ is rose_cpp.mnu)
4. modify the Rose menu file to add the scripts to the main menu; add the following instructions at the end of the menu Tools declaration :

```
menu "Formal"
{
option "&Z Generation   Ctrl + G"
{
RoseScript $SCRIPT_PATH\genez.ebx
}
option "&Theorems Generation  "
{
RoseScript $SCRIPT_PATH\geneth.ebx
}
option "&Base Operations Generation  "
{
RoseScript $SCRIPT_PATH\addop.ebx
}
}
```



Now in the Rose environment, you should have in the ‘‘Tools” menu the choice ‘‘Formal”. From the ‘‘Formal” menu, you should have the options "Z Generation”, "Theorems Generation" and "Base Operations Generation”.

2- Description of the example

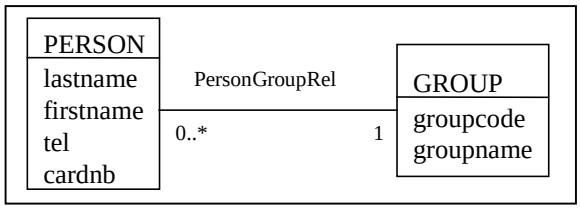
To illustrate the use of RoZ, we will use a simplified variant of an access control system presented in [LED97].

An access control system has to be developed for a company. The access policy can be summarised as follows :

- Every employee of the company has access to a series of buildings of the company.
- The access of a given employee to a given building is granted on the basis of group membership.
- Every employee is member of one (and only one) group and a group may only have access to a restricted set of buildings.

In order to enforce the access control policy, every employee has a unique magnetic card and the doors of the buildings are equipped with card readers. On every entry or exit of a building the employee has to insert his/her card to the reader. If the access is granted, the door is unblocked and the access is logged in a database.

To model this example, we propose the following class diagram. It is composed of 2 classes : “PERSON” and “GROUP”. A person has 4 attributes: his last and first names, the set of his telephone numbers and the number of his magnetic card. Each group is characterised by a name (e.g. “Accounting Department”) and a code (e.g. “AccD”). The PersonGroupRel association links each person to one and only one group.



This model does not express the full specification of the application data. Three additional constraints complement this diagram :

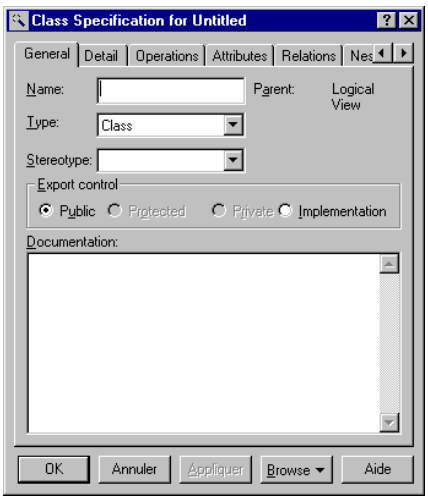
1. Every person has at least one telephone number.
2. The card number is a key of persons.
3. The telephone numbers of the members of a given group have the same prefix.

3- Guided Tour of RoZ

3.1- Construction of the class diagram example in RoZ

The example will be constructed in a class diagram window of Rose. So you should open Rose and a class diagram window. Note that when you open Rose, the main class diagram window is automatically opened.

We will start by creating the classes “PERSON” and “GROUP”. To create a class, you should click on the class button of the toolbox . Then when you click on the class diagram window, an “empty” class is created. To describe the name and the attributes of the class, open the form of the class by choosing “Specification” or “OpenSpecification” in the shortcut menu of the class (put the cursor on the class and click on the right button to see the shortcut menu). You should obtain this window.



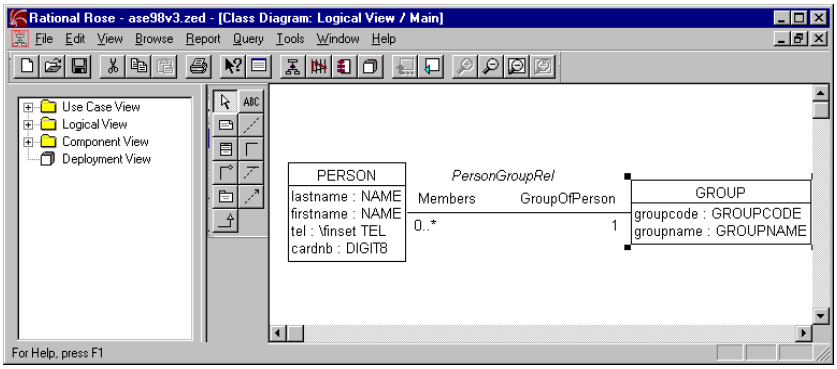
In the “General” tab of the form, enter the name of the class “PERSON” in the name field.

Then to describe the attributes, select the “Attributes” tab. Click on the right button of the mouse and select the “insert” choice. You can enter the name of an attribute (for instance “lastname”). Do the same to enter the name of the other attributes of “PERSON”. To have a complete Z specification, the type of the attributes must be defined. To enter the attribute type, you can open the form of the attribute (put the cursor on the attribute, click on the right button of the mouse and select the “Specification” choice in the shortcut menu). Enter the type of the attribute in the “Type” field of “Class Attribute Specification” form. The type of the attribute must be written as a Z Latex type. For instance, the attribute “tel” is multivaluated (this means that a person can have several telephone numbers). So we specify that its type is a set of telephone numbers by writing “\finset TEL” for the type of “tel”.

Using the previous instructions, you can create the classes “PERSON” and “GROUP” and their attributes.

Now, we will create the association between “PERSON” and “GROUP”. To create an association, you should click on the association relationship button of the toolbox . Then click on the class “PERSON” and letting the button of the mouse pushed, put the cursor on the class “GROUP”. You should have an association between the classes “PERSON” and “GROUP”. Put the cursor on the association, press the right button of the mouse and select the “Specification” choice in the shortcut menu to obtain the form of the association. Then you can enter the name of the association “PersonGroupRel” and the name of its roles : roleA is “Members” while roleB is

"GroupOfPerson". To finish the association description, choose the cardinalities in the shortcut menu of the role. For "Members", click on the right button on the role and select the "Zero and More" choice in the shortcut menu; for "GroupOfPerson" choose "1". Like for attribute type, the role names and the cardinalities are necessary to generate a Z specification without holes.



At this step, we have described the class diagram of the access control system. Now we will add the expression of the constraints. The constraints are expressed in the Z Latex syntax in the "Documentation" field of the forms.

The first constraint (every person has at least one telephone number) carries on the attribute "tel". It is expressed in Z by :

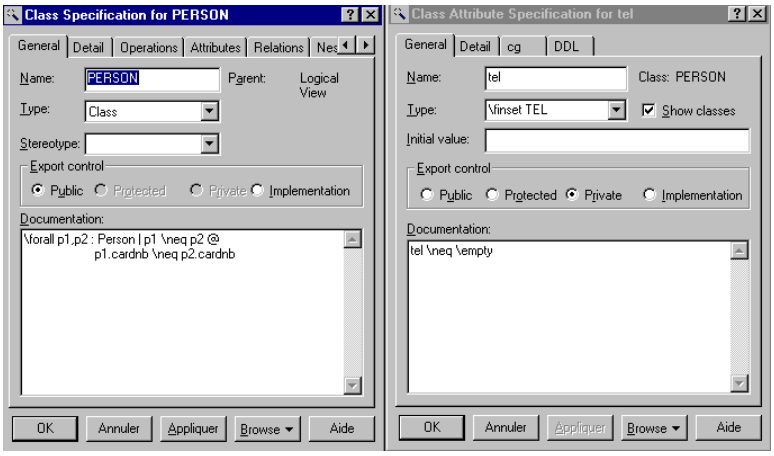
$$tel \neq \varnothing$$

We will write this as a Z Latex type in the field "Documentation" of the attribute "tel". So you have to open the form of "PERSON", to go into the "Attributes" tab and to open the form of "tel". Then write in the "Documentation" field :

```
tel \neq \empty
```

The second constraint (the card number is a key of persons.) means that two different persons have distinct card numbers. It is written in Z :

$$\forall p1,p2 : Person \mid p1 \neq p2 \bullet p1.cardnb \neq p2.cardnb$$



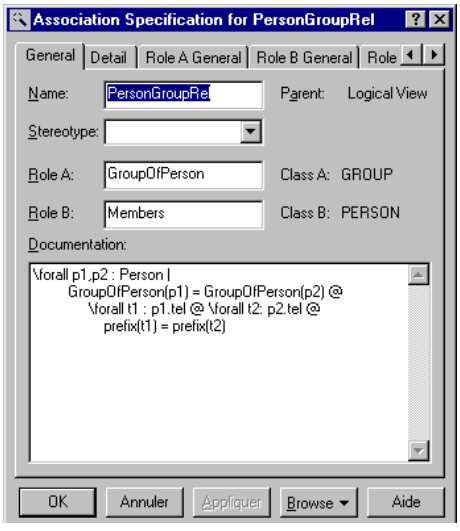
We will write this as a Z Latex type in the field documentation of the class "PERSON". So you have to open the form of "PERSON" and to write in "Documentation" :

```
\forall p1,p2 : Person | p1 \neq p2 @ p1.cardnb \neq p2.cardnb
```

Finally we have to express the third constraint (the telephone numbers of the members of a given group have the same prefix.). This constraints carries on the relationship between "PERSON" and "GROUP", so it is expressed in the "Documentation" field of "PersonGroupRel". Let us consider that we have defined a function prefix which gives the prefix of a telephone number.

The constraint is expressed in Z by :

$$\forall p1,p2 : Person \mid GroupOfPerson(p1) = GroupOfPerson(p2) \bullet \forall t1 : p1.tel \bullet \forall t2 : p2.tel \bullet prefix(t1) = prefix(t2)$$



To write this constraint, open the form of “PersonGroupRel” and write on the “Documentation” field :

\forall p1,p2 : Person |
GroupOfPerson(p1) = GroupOfPerson(p2) @
\forall t1 : p1.tel @ \forall t2 : p2.tel @ prefix(t1) = prefix(t2)

RoZ automatically generates the Z schemas skeletons corresponding to a UML class diagram. To have a complete Z specification, you must add information like the definition of the type of the attributes and the constraints on the class diagram. We have shown how constraints can be included in the RoZ environment.

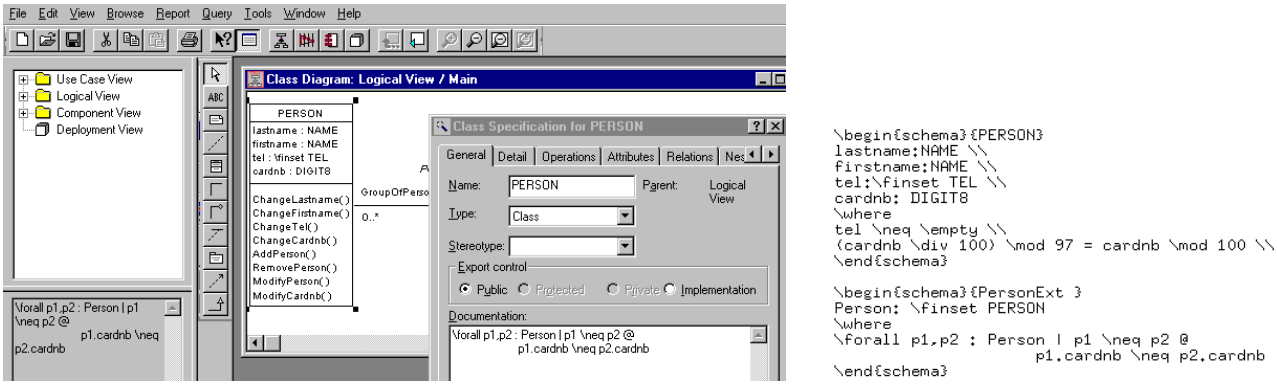
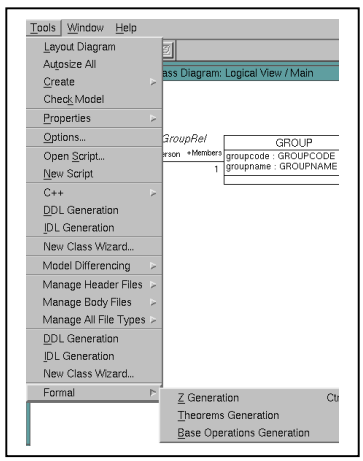


Figure 2 – Generation of a Z specification in Latex format

On the contrary, the type definition cannot be done in the RoZ environment, you have to create a file containing it. The type declaration must be in the Z Latex format. In the example of access control system, we have to define the types NAME, TEL, DIGIT8, GROUPCODE and GROUPNAME. Moreover the third constraint (the telephone numbers of the members of a given group have the same prefix.) requires the declaration of the prefix function. So create a file “ACSTypes.zed” which contains the following “sentences” (Appendix A):

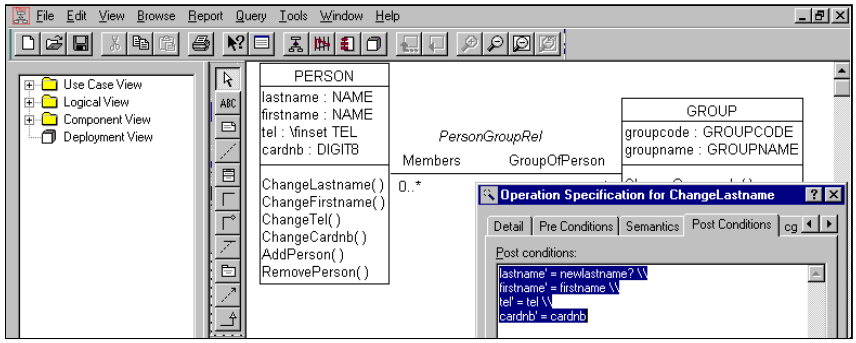
```
\begin{zed}
[NAME,TEL] \also
DIGIT8 == 0 \upto 99999999
\end{zed}
\begin{zed}
[GROUPCODE,GROUPNAME]
\end{zed}
\begin{zed}
[PREFIX]
\end{zed}

\begin{axdef}
prefix : TEL \surj PREFIX
\end{axdef}
```

3.2- Generation of base operations



The generation of base operations is realized by selecting the “Generate base Operations “ from the “Tools” menu and the sub-menu “Formal”.
So from a UML class diagram, RoZ has generated the specification of all the base operations of the diagram classes. For each attribute of a given class, a modification operation is generated. Moreover if the class is concrete, the specification of operations adding or deleting an object to the class objects are added. For the class “PERSON”, the operations “ChangeLastname”, “ChangeFirstname”, “ChangeTel”,

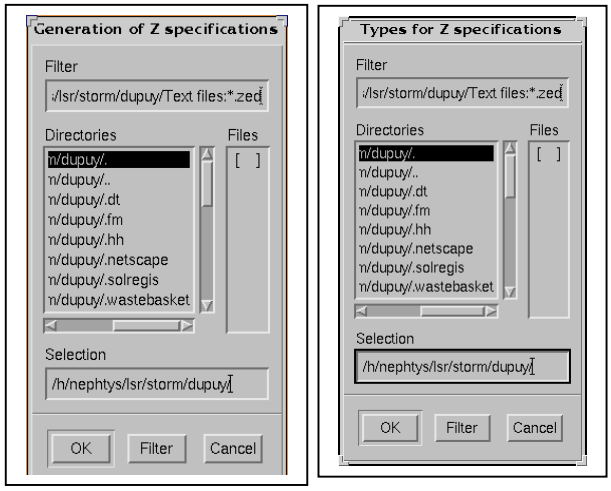
"ChangeCardnb", "AddPerson" and "Remove-Person" are generated automatically. To see the specification of these operations, you can open their form. For instance, let us consider "ChangeLastname" which allows to modify the last name of a person: open the form of "PERSON" by selecting the "Specification" choice in the shortcut menu, go into the "Operations" tab and open the form of "ChangeLastname" by selecting the "Specification" choice in the shortcut menu. You should have a window named "Operation Specification for ChangeLastname". In the "Details" tab, you can see that the operation has an argument "newlastname" which is the input parameter of the operation. The "Semantics" field contains the key-word *intension operation* which signifies that "ChangeLastname" is an operation on the attributes of the class. In the "PostConditions" tab, the post condition of the operation is written as a Z Latex type : it means that the new value of the attribute "lastname" is the argument of the operation "newlastname" and that the others attributes of "PERSON" are unchanged by the operation.



Here the form of the operation "ChangeLastname" shows what type of postcondition is generated for an operation changing an attribute.

3.3- Generation of a Z specification

The automatic generation of a Z specification from a UML class diagram and its additional information is called by selecting the "Z Generation" submenu from the "Formal" menu. The window "Generation of Z specifications" opens. In this window, you should specify the name of the file in which the Z specification will be generated. For the access control system, you can call this file "ACS.zed". When you have enter the name of the file for Z specification and click on the "OK" button, another window ("Types for Z specifications") is opened. Then you have to select the file which contains the type declaration for the generated Z specifications. Here enter "ACSType.zed". Then RoZ generates a file "ACS.zed" which is given in Appendix B.



The "ACS.zed" file is composed of

1. the translation of the class "PERSON"
2. the translation of the operations of "PERSON" : "ChangeLastname", "ChangeFirstname", "ChangeTel", "ChangeCardnb", "AddPerson" and "RemovePerson"
3. the translation of the class "GROUP"
4. the translation of the operations of "GROUP" : "ChangeGroupname", "ChangeGroupcode", "AddGroup" and "RemoveGroup"
5. the translation of the association relationship between "PERSON" and "GROUP"
6. the translation of the global view of the diagram which here only corresponds to the relationship between "PERSON" and "GROUP"

The classes “GROUP” and “PERSON”, and their operations are translated according to the same translation rules. So we will only detail here the translation of class “PERSON”. The static part of “PERSON” is represented by two Z schemas “PERSON” and “PersonExt”. The first one defines attributes and the second one describes the set of existing persons. These schemas contain the constraints expressed in the “Documentation” fields of “PERSON” and “tel”. The constraint on the telephone numbers is expressed in the Z schema defining attributes, while the key constraint is defined in the Z schema representing the existing objects of “PERSON”.

$$
\begin{array}{|l}
\hline
PERSON \\
lastname : NAME \\
firstname : NAME \\
tel : \mathbb{F}\, TEL \\
cardnb : DIGIT8 \\
\hline
tel \neq \varnothing \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
PersonExt \\
Person : \mathbb{F}\, PERSON \\
\hline
\forall\, p1, p2 : Person \mid p1 \neq p2 \bullet p1.cardnb \neq p2.cardnb \\
\hline
\end{array}
$$

There are 2 kinds of base operations which have been generated automatically by RoZ and which will be translated into Z schemas. Their translation depends on their type :

- if an operation is on the attributes of the class (key word “intension operation”), it is translated by a Z schema to express the effects of the operation on the attribute.
  Moreover if the class is concrete, the operation is promoted at the level of the existing objects. To promote an operation, there is an generic-purpose operation “ChangePERSON” which can be combined with all the operation on the attributes of the class. “ChangePERSON” modifies two kinds of variables : “PersonExt” and an instance of the schema “PERSON” corresponding to the input parameter x?. θPERSON is a variable of type “PERSON” and θPERSON’ denotes its final value.

$$
\begin{array}{|l}
\hline
ChangePERSON \\
\Delta PersonExt \\
\Delta PERSON \\
x? : PERSON \\
\hline
x? \in Person \\
\theta PERSON = x? \\
Person' = Person \setminus \{x?\} \cup \{\theta PERSON'\} \\
\hline
\end{array}
$$

  So the specification of the operation “ChangeLastname” which changes the lastname of a person is obtained by a Z schema and a composed operation. “PersonChangeLastname” promotes the “PERSONChangeLastname” at the level of the existing objects. It promotes the change of “PERSONChangeLastname” to a given object x? of “Person”. It also results in the modification of “Person” since the old element x? is replaced in the set by the new value θPERSON’.

$$
\begin{array}{|l}
\hline
PERSONChangeLastname \\
\Delta PERSON \\
newlastname? : NAME \\
\hline
lastname' = newlastname? \\
firstname' = firstname \\
tel' = tel \\
cardnb' = cardnb \\
\hline
\end{array}
$$

$$PersonChangeLastname == ChangePerson \land PERSONChangeLastname$$

- An operation on the existing objects of the class (keyword “extension operation”) gives rise to a Z schema. For instance, “AddPerson” takes a person as input and adds it to the set of the existing persons.

$$
\begin{array}{|l}
\hline
AddPerson \\
\Delta PersonExt \\
person? : PERSON \\
\hline
Person' = Person \cup \{person?\} \\
\hline
\end{array}
$$

The translation of the association relationship “PersonGroupRel” is made by specifying each of its roles in a Z schema. Thus the two roles “Members” and “GroupOfPerson” are specified. Each role is declared by a function : “GroupOfPerson” is a function from “PERSON” to “GROUP” while “Members” links a group to a set of persons. The first constraint on the schema expresses that the domain of “GroupOfPerson” is identical to set “Person”, i.e. every element of the existing persons is a member of a group. The second constraint expresses that the range of this function is a subset of the existing groups, i.e. some groups may have no members. The third constraint expresses “Members” as a set comprehension in terms of “GroupOfPerson” i.e. the members of a group must be linked to that group.

$$
\begin{array}{|l}
\hline
PersonGroupRelRel1 \\
GroupExt;\ PersonExt \\
GroupOfPerson : PERSON \pfun GROUP \\
Members : GROUP \pfun \mathbb{F}\, PERSON \\
\hline
\mathrm{dom}\, GroupOfPerson = Person \\
\mathrm{ran}\, GroupOfPerson \subseteq Group \\
Members = \{x : \mathrm{ran}\, GroupOfPerson \bullet x \mapsto \\
\{y : \mathrm{dom}\, GroupOfPerson \mid GroupOfPerson(y) = x \bullet y\}\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
PersonGroupRelRel \\
PersonGroupRelRel1 \\
\hline
\forall\, p1, p2 : Person \mid \\
GroupOfPerson(p1) = GroupOfPerson(p2) \bullet \\
\forall\, t1 : p1.tel \bullet \forall\, t2 : p2.tel \bullet \\
prefix(t1) = prefix(t2) \\
\hline
\end{array}
$$

The third constraint (all members of a group have the same telephone prefix) holds on this relationship. It is added to the formal specification of “GroupPersonRel” by first including the “GroupPersonRelRel1” schema and then expressing the constraint.
Once each element of a diagram has been expressed in Z, we specify the diagram global view by grouping the Z specification of the relationships (which also include the schemas of the classes). Here we obtain the following Z for the global view of the class diagram.

$$\begin{array}{|l} \textit{GlobalView} \\ \hline \textit{PersonGroupRelRel} \\ \hline \end{array}$$

3.4- Theorems Generation

RoZ automatically generates a theorem to validate proposed pre-conditions for each operation. It uses the information given in the operation form. Let us consider the operation “ChangeTel” which changes the value of the “tel” attribute. The generated specification for this operation is :

$$\begin{array}{|l} \textit{PERSONChangeTel} \\ \Delta PERSON \\ newtel? : \mathbb{F}\ TEL \\ \hline lastname' = lastname \\ firstname' = firstname \\ tel' = newtel? \\ cardnb' = cardnb \\ \hline \end{array}$$

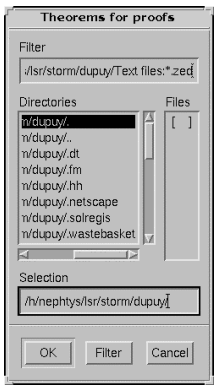
Its actual pre-condition can be composed from the text of the specification and the constrsaints listed in the included schemas. When specifications get complexe, it is good software engineering practice to state these pre-conditions more explicitly. For the “ChengeTel” operation, we identify the pre-condition

$newtel? \neq \varnothing$

In fact, we guess that this pre-condition is strong enough to imply the actual pre-condition of the operation. This predicate states that the new set of telephone numbers must not be empty. We add it to the field “Pre Conditions” of the operation “ChangeTel”. So open the form of the class “PERSON” by putting the cursor on “PERSON” and selecting the “Specification” choice in the shortcut menu). Then go into the “Operation” tab and open the form of “ChangeTel” (put the cursor on “ChangeTel” and select the “Specification” choice). Write the constraint in the “Pre-Conditions” field :

newtel? \neq \empty

Then call the theorems generation by selecting the “Generate Theorems “ from the “Tools” menu and the sub-menu “Formal”. The windows “Theorems for proofs” opens. In this windows, specify the name of the file in which the theorems will be generated. For the example of the access control system, enter the name “ACSthms.zed”. The file “ACSthms.zed” is given in appendix C.



The following theorem is generated for “ChangeTel”:

```
\begin{theorem}{ PERSONChangeTel \_Pre}
        \forall PERSON ;newtel? : \finset TEL |
                newtel? \neq \empty @
                        \pre PERSONChangeTel
\end{theorem}
```

It means that the proposed pre-condition (*newtel? ≠ ∅*) is stronger than the computed pre-condition (*pre PERSONChangeTel*).

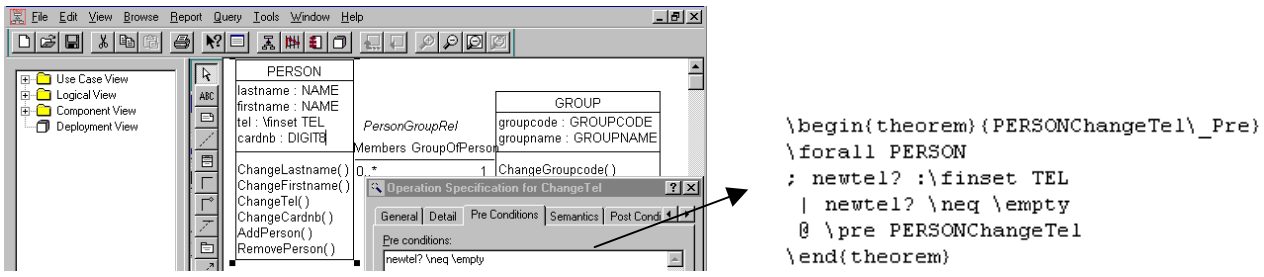


Figure 3 – Generation of theorems

If you want to prove the theorems contained in "ACSthms.zed", you can use the Z-EVES theorem prover. For instance, to prove the theorem "PERSONChangeTel_Pre", you should open Z-EVES, load the file containing the Z specification (type `read "ACS.zed";`) and the file containing the theorems (type `read "ACSthms.zed";`). Then if you want to see what are the theorems to prove, do *print status*; . The pre-condition theorem for "ChangeTel" is prove automatically using the following commands :
try lemma PERSONChangeTel_Pre;
prove by reduce;
which demonstrates that we proposed a correct pre-condition.

4- Principles

4.1- Conditions of use

To use RoZ, you first draw your UML class diagrams with Rose in the main window of "Class diagram : Logical view". Note that **the diagrams should satisfy the following rules** otherwise the Z specification generated will not be correct**:**

- each class attribute must have a type;
- each operation must have at least its signature;
- the association roles and their cardinality should be specified.

Moreover you can annotate the diagrams the additional information for the generation of Z within the form associated to each entity. This information must be described in the Latex format for Z to be able to directly generate a complete Z formal specification.
To obtain a form of an entity :
1- select a given entity
2- click on the right button of the mouse
3- select "Specification" (Windows version) or "Open Specification" (Unix version)
In the obtained form the following fields are used to write formal annotations :

- ❖ the "Documentation" field is always used to express the constraints. The form chosen to express a constraint classifies the constraints and specifies where to include constraints in the formal specification :
  - ➢ Constraints on an attribute and inter-attributes => field "Documentation" of one of the attributes
  - ➢ Constraints on the objects of the class => field "Documentation" of the class
  - ➢ Constraints on an association => field "Documentation" of the association
  - ➢ Constraints specific to an inheritance relationship
    - Constraints on the attributes of the superclass => field "Documentation" of the inheritance relationship
    - Constraints on the objects of the superclass => field "Documentation" of the subclass

- Constraints on the class diagram => field “Documentation” of the class diagram

❖ for an operation, the following fields are used :

- field “Arguments”

It contains the inputs of the operation. The field “Name” contains the name of the inputs and the field “Type” contains their type

- field “return type” or “return class”

This field is used to represent the outputs of the operation. As it can contain only one type, the generated Z operation can have only one output variable. This variable is always named x!.

- field “Semantics”

It is used to define the operation type. It may only contain one of the following keywords :

- *intension operation* for an operation on the attributes of the class
- *extension operation* for an operation on the existing objects of the class
- *composed operation* for an operation composed from other operations
- *promotion operation* for an operation on the attributes and the existing objects of the class
- otherwise the filed “Semantics” contains the explicit declaration part of the operation.

- field “Post Conditions”

It contains the expression of the postcondition of the operation in Z.

- field “Pre Conditions”

It contains an **identified** precondition of the operation in Z for the generation of proof obligations.

Be careful **not introducing** in these fields **information written in another language than Z in the Latex format**.

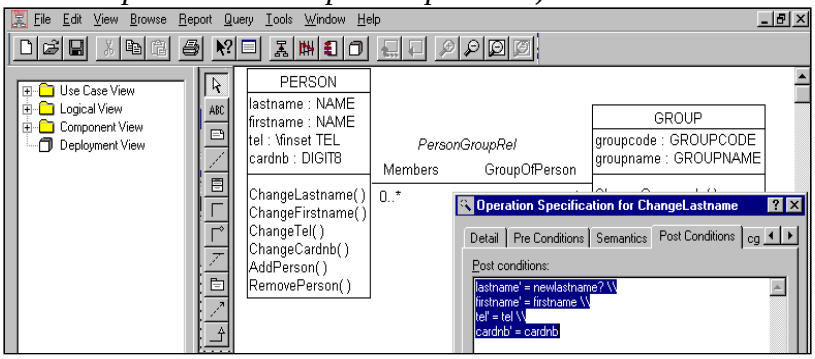
4.2- Generation of base operations

From a UML class diagram, the specification of all the base operations of the diagram classes are generated : for each attribute of a given class, a modification operation is generated. Moreover if the class is concrete, the specification of operations adding or deleting an object to/from the class objects are added.

Thus RoZ automatically generates the specification of these operations in the Z Latex format. The specification generated is composed of the name of the operation, its signature and its post-condition. So for each operation, the following fields of the operation form are fulfilled :

- “Arguments” and “Return Type” for the signature
- “Post Conditions” for the postcondition
- “Semantics” (*intension operation* ,*extension operation* or *composed operation*)

In our example, for the class “PERSON”, the operations “ChangeLastname”, “ChangeFirstname”, “ChangeTel”, “ChangeCardnb”, “AddPerson” and “Remove-Person” are generated automatically. Here the form of the operation “ChangeLastname” shows what kind of postcondition is generated for an operation changing an attribute.



4.3- Z Generation

`Rules to generate Z from a UML class diagram`

Class

A UML class has two aspects : it is an object factory and an object tank. We define the first aspect as the class intension and the second one as the class extension. So two Z schemas are used to describe it. The first one defines attributes and the second one describes the set of existing objects. For instance, the class “PERSON” is translated into the following Z schemas:

$$
\begin{array}{|l}
\hline
PERSON \\
\hline
lastname : NAME \\
firstname : NAME \\
tel : \mathbb{F}\, TEL \\
cardnb : DIGIT8 \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
PersonExt \\
\hline
Person : \mathbb{F}\, PERSON \\
\hline
\end{array}
$$

Operations and their promotions

The translation of an operation depends on its type :

- if an operation is on the class intension (key word “intension operation”), it is translated by a Z schema to express the effects of the operation on the attribute.

  Moreover if the class is concrete, the operation is promoted at the level of the existing objects. To promote an operation, there is an generic-purpose operation “ChangeCLASS” which can be combined with all operations on the attributes of the class. For instance, the class “PERSON” has the operation “ChangePERSON”. “ChangePERSON” modifies two kinds of variables : “PersonExt” and an instance of the schema “PERSON” corresponding to the input parameter x?. θPERSON is a variable of type “PERSON” and θPERSON’ denotes its final value.

$$
\begin{array}{|l}
\hline
ChangePERSON \\
\Delta PersonExt \\
\Delta PERSON \\
x? : PERSON \\
\hline
x? \in Person \\
\theta PERSON = x? \\
Person' = Person \setminus \{x?\} \cup \{\theta PERSON'\} \\
\hline
\end{array}
$$

  So the specification of the operation “ChangeLastname” which changes the lastname of a person is obtained by a Z schema and a composed operation. “PersonChangeLastname” promotes the “PERSONChangeLastname” at the level of class extension. It promotes the change of “PERSONChangeLastname” to a given object x? of “Person”. It also results in the modification of “Person” since the old element x? is replaced in the set by the new value θPERSON’.

$$
\begin{array}{|l}
\textit{PERSONChangeLastname} \\
\Delta PERSON \\
newlastname? : NAME \\
\hline
lastname' = newlastname? \\
firstname' = firstname \\
tel' = tel \\
cardnb' = cardnb \\
\hline
\end{array}
$$

$$PersonChangeLastname == ChangePerson \land PERSONChangeLastname$$

An operation on the class extension (keyword "extension operation") gives rise to a single Z schema. For example, "AddPerson" takes a person as input and adds it to the set "Person".

$$
\begin{array}{|l}
\textit{AddPerson} \\
\Delta PersonExt \\
person? : PERSON \\
\hline
Person' = Person \cup \{person?\} \\
\hline
\end{array}
$$

- Operation can also be obtained by composing other operations. Then this kind of operation is identified in RoZ by the key-word "composed operation" and the composition should be given explicitly in the "Post Condition" field.

Association

Associations link two or more classes non necessarily distinct. To represent an association in Z, we choose to specify each of its roles. For instance, for the association "PersonGroupRel", the two roles "Members" and "GroupOfPerson" are specified. Each role is declared by a function. Then the predicates specify that the association links existing objects of the classes (definition of the domain and codomain of the first role). The second role is expressed by the way of the first one:

$$
\begin{array}{|l}
\textit{PersonGroupRel1} \\
GroupExt;\ PersonExt \\
Members : PERSON \pfun GROUP \\
GroupOfPerson : GROUP \pfun \mathbb{F}\, PERSON \\
\hline
\mathrm{dom}\, Members = Person \\
\mathrm{ran}\, Members \subseteq Group \\
GroupOfPerson = \{x : \mathrm{ran}\, Members \bullet x \mapsto \\
\{y : \mathrm{dom}\, Members \mid Members(y) = x \bullet y\}\} \\
\hline
\end{array}
$$

Simple Inheritance

Let us consider that the class "PERSON" can be specialised in 2 classes "STUDENT" and "TEACHER". "STUDENT" has for specific attributes the student number and the sports practised by the student.

First, each superclass gives rise to a Z schema defining its attributes. For "PERSON", we have the Z schema "PERSONATTR".

$$
\begin{array}{|l}
\hline
PERSONATTR \\
\hline
lastname : NAME \\
firstname : NAME \\
tel : \mathbb{F}\, TEL \\
cardnb : DIGIT8 \\
\hline
\end{array}
$$

Second, a superclass is defined as the disjunction of its subclasses. A person can be a student or a teacher :

$$PERSON == TEACHER \vee STUDENT$$

Like for other class, the set of the existing objects of the superclass is defined. A schema "PersonExt" is created and has for variable "Person" the set of the existing persons of the database. But as an object of the superclass can also be an object of one of its subclasses, "PersonExt" has two other variables : "Teacher2" (resp. "Student2") which represents the set of the existing persons who are also teachers (resp. students). The predicate part is used to express the link between these set of objects : the set "Teacher2" (resp. "Student2") is a subset of "Person". Moreover, we add a predicate to express that each existing person who is also a teacher (resp. a student) also exists in "TEACHER" (resp. "STUDENT") : for all element of "Teacher2" (resp."Student2"), there is a corresponding element of "TEACHER" (resp "STUDENT").

$$
\begin{array}{|l}
\hline
PersonExt \\
\hline
Person : \mathbb{F}\, PERSON \\
Teacher2 : \mathbb{F}\, PERSON \\
Student2 : \mathbb{F}\, PERSON \\
\hline
Teacher2 \subseteq Person \\
\forall\, x : Teacher2 \bullet \exists\, y : TEACHER \bullet \\
x.lastname = y.lastname \wedge x.firstname = y.firstname \wedge \\
x.tel = y.tel \wedge x.cardnb = y.cardnb \wedge x.nbh = y.nbh \\
Student2 \subseteq Person \\
\forall\, x : Student2 \bullet \exists\, y : STUDENT \bullet \\
x.lastname = y.lastname \wedge x.firstname = y.firstname \wedge \\
x.tel = y.tel \wedge x.cardnb = y.cardnb \wedge \\
x.stnum = y.stnum \wedge x.sport = y.sport \\
\hline
\end{array}
$$

Like superclass, each subclass gives rise to a Z schema defining its specific attributes. For the "STUDENT" class, we have the Z schema "STUDENTATTR". The description of all the attribute of a subclass is obtained by including the schema of its superclass attributes and its attribute schema. The schema "STUDENT" includes the schemas "PERSONATTR" (containing the definition of the attributes of "PERSON") and "STUDENTATTR. So an element of "STUDENT" has the attributes of "PERSON" and those of "STUDENT"

$$
\begin{array}{|l}
\hline
STUDENTATTR \\
\hline
stnum : STNUM \\
sport : \mathbb{F}\, SPORT \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
STUDENT \\
\hline
PERSONATTR \\
STUDENTATTR \\
\hline
\end{array}
$$

For each subclass, the set of its existing objects is defined by using another Z schema. The predicate part of the schema is used to express that for each existing element of the subclass, there

exists an element of its superclass. For each student in the set “Student”, there is a corresponding person in the set of the existing persons which are also student “Student2”.

$$
\begin{array}{|l}
\hline
StudentExt \\
PersonExt \\
Student : \mathbb{F}\, STUDENT \\
\hline
Student = \{x : STUDENT \mid \exists\, y : Student2 \bullet \\
x.lastname = y.lastname \land x.firstname = y.firstname \land \\
x.tel = y.tel \land x.cardnb = y.cardnb \land \\
x.stnum = y.stnum \land x.sport = y.sport\} \\
\hline
\end{array}
$$

The translation of the operation of a subclass follows the rules described above. To simulate the inheritance of the superclass operations, we import them at the extension level of the subclass. As for other classes, For each subclass, there is a schema “ChangeClass” which allows to promote operations at the extension level of the subclass. Then this operation can be composed with all the operations on the attributes of the superclass. For the subclass “STUDENT”, there is a schema “ChangeStudent”. The following example illustrates how to compose “ChangeStudent” to import the operation “PERSONChangeLastname” at the extension level of “STUDENT”.

$$
\begin{array}{|l}
\hline
ChangeStudent \\
\Delta StudentExt \\
\Delta STUDENT \\
x? : STUDENT \\
\hline
x? \in Student \\
\theta STUDENT = x? \\
Student' = Student \setminus \{x?\} \cup \{\theta STUDENT'\} \\
\hline
\end{array}
$$

$$
StudentChangeLastname == \\
\quad ChangeStudent \land PERSONChangeLastname
$$

Aggregation/Composition

The aggregation relationship is considered in UML like a particular association which expresses a weak ownership. So we propose to translate in Z, an aggregation like an association which contains “Aggreg” in the name of its Z schema.

The composition relationship expresses a strong ownership. We propose to translate it in Z like an association which contains “Comp” in the name of its Z schema. A proof obligation could also be generated but at the moment, the tool does not do it.

View of the database

Once all the classes and theirs associations have been specified, they must be grouped to obtain a global view of the model. So a new Z schema is used : it includes all the schemas of associations.

$$
\begin{array}{|l}
\hline
GlobalView \\
PersonGroupRel \\
\hline
\end{array}
$$

`Naming conventions`

The translation of UML specifications uses the following naming conventions to produce the Z specifications :

- **class :** the name of the intension schema is in capital letters while the name of the extension one is a composition of the class name and "Ext". The name of the set of the existing objects is the name of the class with its first letter in capital letters and the other ones in small letters.
- **attribute:** its name is in small letters.
- **operation:** the name of an operation on the attributes is composed at the intension level by the name of the class in capital letters and by the name of the operation; the name of the operation at the extension level is composed by the name of the class with the first letter in capital and by the name of the operation. The other operations keep their name in Z.
- **inheritance:** the translation of inheritance follows the naming convention for classes, attributes and operations. It also has two particularities: the name of the schema containing the specific attributes of a super or a sub-class is composed by the name of the class in capital letters and "ATTR"; the set of the existing objects of a subclass declared in the extension schema of the superclass has the name of the subclass with its first letter in capital letters, the other ones in small letters and a 2 to terminate it.
- **association/aggregation/composition**: the name of the Z schema representing a relationship is of the name of the relationship plus the type of the relationship. For a relationship without particularity, the name of its Z schema is composed with the name of the relationship and "Rel". If the relationship is an aggregation, the name of its Z schema terminates with "Aggreg"; if it is a composition, it ends with "Comp".

External annotations

Following the rules described above, RoZ will automatically generate the corresponding Z schemas skeletons. To have a complete Z specification, you must add information like the definition of the type of the attributes and the constraints on the class diagram.
The type definitions cannot be done in the RoZ environment, you have to create a file containing it. The type declarations must be in the Z Latex format.
The constraints on the diagram are expressed in the field "Documentation" of the form attached to an object. And the specification of an operation is expressed in the form of this operation.
The "Conditions of Use" (section 4.1) paragraph summarizes where each type of information must be expressed in the environment.

2.5- Theorems Generation

RoZ generates theorems to verify that a proposed pre-condition is actually a pre-condition of the operation. So for each operation, a theorem of the following form is generated ([LED98]):

$\forall State, i? : IN \mid precondition(State,i?)$ **pre** *Op*

Where Op is the operation
State is the schema of the operation
i? represents the inputs parameters of the operation
precondition(State,i?) is a predicate representing a pre-condition of the operation (not necessarily the weakest pre-condition)
pre is is the Z operator which extracts the weakest pre-condition of the operation

The theorems can then be proved using a theorem prover that supports the Z Latex syntax.

References

[DUP99] S.Dupuy, On considering constraints in UML, Rapport technique Laboratoire LSR-PFL

[LED97] Y. Ledru, Etude de cas : Système de contrôle d'accès (version 1.1), Avril 1997 http://www.lsr.imag.fr/LesPersonnes/YvesLedru/Contrôle_Acces/

[LED98] Y.Ledru, Identifying pre-conditions with the Z/EVES theorem prover, Dans Int. IEEE Conference on Automated Software Engineering'98, October 1998

[ROSE4.0] Rational Software Corporation, Rational Rose – Using Rational Rose 4.0, 1996

[ROSE98] Rational Software Corporation, Rational Rose – Using Rational Rose 98, 1998

[SAA97] M. Saaltink, The Z/EVES system, Proc. 10th Int. Conf. On the Z formal method (ZUM), Reading, UK, April 1997

[SPI95] J.M. Spivey, La notation Z, Prentice-Hall International, 1995

[UML97] Rational Software Corporation, UNIFIED MODELING LANGUAGE – notation guide version 1.1, Septembre1997

Appendix A

Type declarations
(file ACSTypes.zed)

```
\begin{zed}
[NAME,TEL] \also
DIGIT8 == 0 \upto 99999999
\end{zed}

\begin{zed}
[GROUPCODE,GROUPNAME]
\end{zed}

\begin{zed}
[PREFIX]
\end{zed}

\begin{axdef}
prefix : TEL \surj PREFIX
\end{axdef}
```

Appendix B

Z specification as generated by RoZ
(file ACS.zed)

```
\documentstyle[oz,z-eves]{article}

\begin{document}

\include{/h/nephtys/lsr/storm/dupuy/THESE/ROSE/demo/ACSType.zed}

\begin{schema}{PERSON}
lastname:NAME \\
firstname:NAME \\
tel:\finset TEL \\
cardnb:DIGIT8 \\
\where
tel \neq \empty \\
\end{schema}

\begin{schema}{PersonExt }
Person: \finset PERSON \\
\where
\forall p1,p2 : Person | p1 \neq p2 @
                    p1.cardnb \neq p2.cardnb
\end{schema}

\begin{schema}{ChangePerson}
\Delta PersonExt \\
\Delta PERSON \\
x? : PERSON
\where
x? \in Person \\
\theta PERSON = x? \\
Person' = Person \setminus \{x?\} \cup \{\theta PERSON'\}
\end{schema}

\begin{schema}{PERSONChangeLastname}
\Delta PERSON \\
newlastname? :NAME \\
\where
lastname' = newlastname? \\
firstname' = firstname \\
tel' = tel \\
cardnb' = cardnb
\end{schema}

\begin{zed}
PersonChangeLastname \defs ChangePerson \land PERSONChangeLastname
\end{zed}

\begin{schema}{PERSONChangeFirstname}
\Delta PERSON \\
newfirstname? :NAME \\
\where
lastname = lastname' \\
firstname = newfirstname? \\
tel' = tel \\
cardnb' = cardnb
\end{schema}
```

```
\begin{zed}
PersonChangeFirstname \defs ChangePerson \land PERSONChangeFirstname
\end{zed}

\begin{schema}{PERSONChangeTel}
\Delta PERSON \\
newtel? :\finset TEL \\
\where
lastname’ = lastname \\
firstname’ = firstname \\
tel’ = newtel? \\
cardnb’ = cardnb
\end{schema}

\begin{zed}
PersonChangeTel \defs ChangePerson \land PERSONChangeTel
\end{zed}

\begin{schema}{PERSONChangeCardnb}
\Delta PERSON \\
newcardnb? :DIGIT8 \\
\where
lastname’ =lastname \\
firstname’ = firstname \\
tel’ = tel \\
cardnb’ = newcardnb?
\end{schema}

\begin{zed}
PersonChangeCardnb \defs ChangePerson \land PERSONChangeCardnb
\end{zed}

\begin{schema}{AddPerson}
\Delta PersonExt  \\
person? :PERSON \\
\where
Person’ = Person \cup \{person? \}
\end{schema}

\begin{schema}{RemovePerson}
\Delta PersonExt  \\
person? :PERSON \\
\where
Person’ = Person \setminus \{ person? \}
\end{schema}

\begin{schema}{GROUP}
groupcode:GROUPCODE \\
groupname:GROUPNAME \\
\end{schema}

\begin{schema}{GroupExt }
Group: \finset GROUP \\
\where
\forall g1,g2 : Group | g1 \neq g2 @
                      g1.groupcode \neq g2.groupcode
           \land g1.groupname \neq g2.groupname
\end{schema}
```

```
\begin{schema}{ChangeGroup}
\Delta GroupExt \\
\Delta GROUP \\
x? : GROUP
\where
x? \in Group \\
\theta GROUP = x? \\
Group' = Group \setminus \{x?\} \cup \{\theta GROUP'\}
\end{schema}

\begin{schema}{GROUPChangeGroupcode}
\Delta GROUP \\
newgroupcode? :GROUPCODE \\
\where
groupcode' = newgroupcode? \\
groupname' = groupname
\end{schema}

\begin{zed}
GroupChangeGroupcode \defs ChangeGroup \land GROUPChangeGroupcode
\end{zed}

\begin{schema}{GROUPChangeGroupname}
\Delta GROUP \\
newgroupname? :GROUPNAME \\
\where
groupname' = newgroupname? \\
groupcode' = groupcode
\end{schema}

\begin{zed}
GroupChangeGroupname \defs ChangeGroup \land GROUPChangeGroupname
\end{zed}

\begin{schema}{AddGroup}
\Delta GroupExt  \\
group? :GROUP \\
\end{schema}

\begin{schema}{RemoveGroup}
\Delta GroupExt  \\
group? :GROUP \\
\where
Group' = Group \setminus \{ group? \}
\end{schema}

\begin{schema}{PersonGroupRelRel1}
GroupExt ; PersonExt \\
Members : PERSON \pfun  GROUP \\
GroupOfPerson : GROUP \pfun  \finset PERSON
\where
\dom Members = Person \\
\ran Members \subseteq Group \\
GroupOfPerson = \{ x : \ran Members @ x \mapsto
    \{ y : \dom Members | Members(y) = x @ y \} \}
\end{schema}
```

```
\begin{schema}{PersonGroupRelRel}
PersonGroupRelRel1
\where
\forall p1,p2 : Person |
         GroupOfPerson(p1) = GroupOfPerson(p2) @
              \forall t1 : p1.tel @ \forall t2: p2.tel @
                  prefix(t1) = prefix(t2)
\end{schema}

\begin{schema}{GlobalView}
PersonGroupRelRel \\
\end{schema}
\end{document}
```

Appendix C

Proof obligations as generated by RoZ (file ACSthms.zed)

```
\begin{theorem}{PERSONChangeLastname\_Pre}
\forall PERSON
; newlastname? :NAME
 @ \pre PERSONChangeLastname
\end{theorem}

\begin{theorem}{PERSONChangeFirstname\_Pre}
\forall PERSON
; newfirstname? :NAME
 @ \pre PERSONChangeFirstname
\end{theorem}

\begin{theorem}{PERSONChangeTel\_Pre}
\forall PERSON
; newtel? :\finset TEL
 | newtel? \neq \empty
 @ \pre PERSONChangeTel
\end{theorem}

\begin{theorem}{PERSONChangeCardnb\_Pre}
\forall PERSON
; newcardnb? :DIGIT8
@ \pre PERSONChangeCardnb
\end{theorem}

\begin{theorem}{AddPerson\_Pre}
\forall PersonExt
; person? :PERSON
 | \forall p : Person @ person?.cardnb \neq p.cardnb
 @ \pre AddPerson
\end{theorem}

\begin{theorem}{RemovePerson\_Pre}
\forall PersonExt
; person? :PERSON
 @ \pre RemovePerson
\end{theorem}

\begin{theorem}{GROUPChangeGroupcode\_Pre}
\forall GROUP
; newgroupcode? :GROUPCODE
 @ \pre GROUPChangeGroupcode
\end{theorem}

\begin{theorem}{GROUPChangeGroupname\_Pre}
\forall GROUP
; newgroupname? :GROUPNAME
 @ \pre GROUPChangeGroupname
\end{theorem}

\begin{theorem}{AddGroup\_Pre}
\forall GroupExt
; group? :GROUP
 | \forall g : GROUP @ group?.groupcode \neq g.groupcode
     \land group?.groupname \neq g.groupname
```

```
 @ \pre AddGroup
\end{theorem}

\begin{theorem}{RemoveGroup\_Pre}
\forall GroupExt
; group? :GROUP
 @ \pre RemoveGroup
\end{theorem}
```